Computers Math. Applic. Vol. 20, No. 1, pp. 39–42, 1990
Printed in Great Britain.

0097-4943/90 $3.00 + 0.00

A PERMUTATION-GENERATING ALGORITHM

C. ZHOU†
Tsinghua University, Peking, People's Republic of China

(*Received* 27 *February* 1989; *in revised form* 30 *May* 1989)

Abstract—This paper gives an algorithm generating all elements of the symmetric group S_n ordered as a sequence of cosets of S_n/G, where G is a subgroup of order $2n$ (for $n \geqslant 3$) generated by cyclic permutation $C: i \to i+1 \pmod n$ and the reversal permutation $R: i \to n+1-i$, i.e. $\{R^jC^i\}$, where $0 \leqslant j \leqslant 1$ and $0 \leqslant i \leqslant n-1$.

1. THE ALGORITHM OF M. B. WELLS

The author's algorithm is based on the Wells' algorithm whose point is the use of the concept of a number under a permutation in an "active state".

First of all, let's define the concept of a number under a permutation in an "active state". For example, we put an arrow "←" above every number under the permutation 1234. The permutation 1234 is changed into the permutation $\overleftarrow{1}\overleftarrow{2}\overleftarrow{3}\overleftarrow{4}$. We call a number under a permutation in an "active state" when the adjacent number in the arrow direction of the number is smaller than the number itself. Under permutation $\overleftarrow{1}\overleftarrow{2}\overleftarrow{3}\overleftarrow{4}$, the numbers $\overleftarrow{2}$, $\overleftarrow{3}$ and $\overleftarrow{4}$ are in an "active state".

Let P_n express a permutation of length n.

The Wells' algorithm which generates all the permutations from the permutation $P_n = \overleftarrow{1}\overleftarrow{2}\ldots\overleftarrow{n}$ is as follows:

(s1) If there is no number under the permutation P_n in an "active state" then stop, else go to step (s2).
(s2) Find the maximum number under the permutation P_n in an "active state", and let it be M. Transpose M and the adjacent number in the arrow direction of M, and go to step (s3).
(s3) Change the arrow direction of all the numbers under the permutation P_n which are bigger than M, and go to step (s1).

2. THE AUTHOR'S ALGORITHM

S_n is grouped into $(n-1)!/2$ cosets. Each coset consists of all elements obtained from σ, where $\sigma \in S_n$ and is generated in the method similar to that of the Wells' algorithm, by cyclic permutation $C: i \to i+1 \pmod n$ and reversal permutation $R: i \to n+1-i$, i.e. $(R^jC^i \cdot \sigma)$, where $0 \leqslant j \leqslant 1$ and $0 \leqslant i \leqslant n-1$.

The algorithm which generates all the permutations from the permutation $P_n = P_{n-1}n = \overleftarrow{1}\overleftarrow{2}\ldots\overleftarrow{(n-1)}n$ is as follows:

(s1) $P'_n \Leftarrow P_n$.
(s2) Generate $R \cdot P'_n$. If we have had the operation C on the permutation P'_n for $n-1$ times, then go to step (s4), else go to step (s3).
(s3) Generate $C \cdot P'_n$. $P'_n \Leftarrow C \cdot P'_n$, and go to step (s2).
(s4) Find the maximum number under the permutation p_{n-1} in an "active state", and let it be M. If M is 2, then stop, else transpose M and the adjacent number in the arrow direction of M and generate P_n. Go to step (s5).
(s5) Change the arrow direction of the numbers under the permutation P_{n-1} which are bigger than M, and go to step (s1).

Note that the operation C and R are implemented functionally (not really) by increasing n storage locations, which is very important for this algorithm.

†Please address correspondence to Mr C. Zhou, c/o Pu-nan Zhou, Beijing Simulation Center, Box 142-21, Beijing, People's Republic of China.

Before this algorithm generates the first permutation $a[1]a[2]\ldots a[n]$ of S_n, elements $a[1]$, $a[2], \ldots, a[2n]$ are initialized with numbers, where $a[i] = a[i+n]$ and $1 \leqslant i \leqslant n$. When generating a permutation in the method similar to that of Wells', besides doing the necessary operations of Wells', this algorithm needs to transpose a pair of extra adjacent numbers and give a head point to the element $a[1]$ and a tail point to the element $a[n]$. Every time when generating a permutation by the operation C, only shift the head point and the tail point to the right adjacent number, respectively, thus the two points determinate a permutation. Every time when generating a permutation by the operation R, do nothing but see the permutation from right to left.

Now we analyze this algorithm. Compared with the Wells' algorithm, this algorithm uses n storage locations more. We will mainly analyze the time. Assume steps (s1)–(s5) costs time, T_1, T_2, T_3, T_4 and T_5, respectively. We will count the number of times each step is executed as follows:

Step	Number of times
(s1)	$(n-1)_\uparrow/2$
(s2)	$n_\uparrow/2$
(s3)	$(n-1)(n-1)_\uparrow/2$
(s4)	$(n-1)_\uparrow/2$
(s5)	$(n-1)_\uparrow/2$

Thus the average time cost by generating a permutation as $n \to \infty$ is

$$((n-1)_\uparrow/2 \cdot T_1 + n_\uparrow/2 \cdot T_2 + (n-1)(n-1)_\uparrow/2 \cdot T_3 + (n-1)_\uparrow/2 \cdot T_4 + (n-1)_\uparrow/2 \cdot T_5)/n! = (T_2 + T_3)/2.$$

Now we prove that this algorithm is correct.

Lemma 1

$C^i R = RC^{n-i}$, where $0 \leqslant i \leqslant n-1$.

Proof. By mathematical induction on i.

1. When $i = 0$, the statement is true.

2. Assume the statement is true for $i = k$, where $0 < k < n-1$, i.e. $C^k R = RC^{n-k}$, while according to the definition of C, we know that $\forall \sigma \in S_n$, we have $C\sigma C = \sigma$, thus $RC^{n-(k+1)} = C \cdot RC^{(n-k+1)} \cdot C = C \cdot RC^{n-k} = C^{k+1}R$. So when $i = k+1$, the statement is true.

Lemma 2

Let $G = \{R^j C^i \mid 0 \leqslant j \leqslant 1,\ 0 \leqslant i \leqslant n-1\}$, G is a subgroup of S_n.

Proof.

1. $\forall R^{j_1}C^{i_1}$, $R^{j_2}C^{i_2} \in G$, where $0 \leqslant j_1, j_2 \leqslant 1$ and $0 \leqslant i_1, i_2 \leqslant n-1$.

When $j_2 = 0$, we have $R^{j_1}C^{i_1} \cdot R^{j_2}C^{i_2} = R^{j_1}C^{i} \in G$, where $i = i_1 + i_2 \pmod n$.

When $j_2 = 1$, according to Lemma 1, we have $R^{j_1}C^{i_1} \cdot R^{j_2}C^{i_2} = R^{j_1+1}C^{i} \in G$, where $i = n - i_1 + i_2 \pmod n$.

2. $\forall R^j C^i \in G$, where $0 \leqslant j \leqslant 1$ and $0 \leqslant i \leqslant n-1$. According to Lemma 1, we have $(R^jC^i)^{-1} = C^{n-i}R^j$.

When $j = 0$, $C^{n-i}R^j = C^{n-i} \in G$.

When $j = 1$, $C^{n-i}R^j = RC^i \in G$.

3. Identity $R^0C^0 \in G$.

It follows that G is a subgroup of S_n.

According to this algorithm, step (s1) generates all the permutations which have the characteristic that n is the rightest number of permutations and 1 is on the left-hand side of 2. Thus step (s1) can generate $(n-1)_\uparrow/2$ permutations, which we let be a_i, where $1 \leqslant i \leqslant (n-1)_\uparrow/2$.

Lemma 3

$\{Ga_i \mid 1 \leqslant i \leqslant (n-1)_\uparrow/2\}$ is a set of right cosets of S_n.

Proof. Since a_i (a_j) is the only permutation of Ga_i(Ga_j) whose rightest number is n, while $a_i \neq a_j$ ($i \neq j$), thus $Ga_i \cap Ga_j = \phi$, where $1 \leqslant i, j \leqslant (n-1)_\uparrow/2$. Since $|Ga_i| = 2n$, thus

$$\sum_{i=1}^{(n-1)!/2} |Ga_i| = n_\uparrow.$$

So $\{Ga_i \mid 1 \leqslant i \leqslant (n-1)_\uparrow/2\}$ is a set of right cosets of S_n.

Theorem 1

$\{Ga_i | 1 \leqslant i \leqslant (n-1)_\uparrow/2\}$ is a partition of S_n.

Proof. According to Lemma 3, we have $\{Ga_i | 1 \leqslant i \leqslant (n-1)_\uparrow/2\}$ is a partition of S_n.

In fact Theorem 1 has proved that this algorithm is right.

3. CONCLUSION

(1) Theoretically, the average time of generating a permutation (as $n \to \infty$) for this algorithm is half of the time cost by judging the number of times the operation C is executed and shifting two points, while for the Wells' is the time cost by finding the two adjacent numbers, transposing them and changing the parameters describing the state. Empirically, the Wells' algorithm costs three times more time than this one to generate all the permutations of S_n (when $n = 7$–10). In respect of the time, I think this algorithm is better than the one that Wells proposed. Of all the permutation-generating algorithms, Wells' is good.

(2) For some problems, it might be enough to only consider the representive elements of each coset of S_n. Changing this algorithm slightly will meet the demand.

REFERENCES

1. K. C. Lu, *The Algorithm and Analyses of Combinatorial Mathematics*, Vol. 1. Tsinghua University, Beijing, China (1983).
2. H. S. Wilf, *Combinatorial Algorithms*, Vol. 7. Academic Press, New York (1978).
3. M. B. Wells, *Math. Comput.* **15**, 192–195 (1961).

APPENDIX

The implementation of the Wells' algorithm (using Algol):

```
procedure pergen(n)
integer n;
begin
  integer p, q, r, i, k;
  integer array a[1: n], e, d[2: n];
  a[1] := 1;
  for i := 2 step 1 until n do
    begin
      a[i] := d[i] := i; e[i] := -1
    end
    sw: q := 0; print a;
    for k := n step -1 until 2 do
      begin
        d[k] := p := d[k] + e[k];
        if p = k then e[k] := -1 /* handling at the right end */
          else if p = 0 then begin
                               e[k] := 1; q = q + 1 /* handling at the left end*/
                             end
                        else begin
                               p := p + q; r := a[p];
                               a[p] := a[p + 1]; a[p + 1] := r;
                               goto sw
                             end
      end
end
```

The implementing procedure of the Wells' algorithm ($n = 4$)

								Direction					
a_1	a_2	a_3	a_4	d_2	d_3	d_4	q	2	3	4	e_2	e_3	e_4
1	2	3	4	2	3	4	0	←	←	←	−1	−1	−1
1	2	4	3	2	3	3	0						
1	4	2	3	2	3	2	0						
4	1	2	3	2	3	1	1						
4	1	3	2	2	2	0	0	←	←	→	−1	−1	1
1	4	3	2	2	2	1	0						
1	3	4	2	2	2	2	0						
1	3	2	4	2	2	3	0						

—continued overleaf

The implementing procedure of the Wells' algorithm ($n = 4$)—*continued*

								Direction					
a_1	a_2	a_3	a_4	d_2	d_3	d_4	q	2	3	4	e_2	e_3	e_4
3	1	2	4	2	1	4	0	←	←	←	−1	−1	−1
3	1	4	2	2	1	3	0						
3	4	1	2	2	1	2	0						
4	3	1	2	2	1	1	0						
4	3	2	1	1	0	0	2	←	→	→	−1	1	1
3	4	2	1	1	0	1	0						
3	2	4	1	1	0	2	0						
3	2	1	4	1	0	3	0						
2	3	1	4	1	1	4	0	←	→	←	−1	1	−1
2	3	4	1	1	1	3	0						
2	4	3	1	1	1	2	0						
4	2	3	1	1	1	1	0						
4	2	1	3	1	2	0	1	←	→	→	−1	1	1
2	4	1	3	1	2	1	0						
2	1	4	3	1	2	2	0						
2	1	3	4	1	2	3	0						

The implementation of the author's algorithm (using Algol);

```
procedure pergen(n)
integer n;
begin
  integer p, q, r, i, k, t;
  integer array a[1: 2n], e, d[3: n − 1];
  for i := 1 step 1 until n do a[i] := a[i + n] := i;
  for i := 3 step 1 until n − 1 do e[i] := −1
sw: q := 0; t := n − 1;
    for k := 1 step 1 until n do
    begin
      t := t + 1;
      for i := k step 1 until t do print a[i]; /* cycling operation */
      for i := t step −1 until k do print a[i] /* reversing operation */
    end
    for k := n − 1 step −1 until 3 do
      begin
        d[k] := p := d[k] + e[k];
        if p = k then e[k] := −1 /* handling at the right end */
          else if p = 0 then begin
                               e[k] := 1; q = q + 1 /* handling at the left end */
                             end
               else begin
                      p := p + q; r := a[p];
                      a[p] := a[p + n] := a[p + 1]; a[p + 1] := a[p + 1 + n] := r;
                      goto sw
                    end
      end
  end
```

The implementing procedure of the author's algorithm ($n = 4$):

										Direction		
a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	d_3	q	3	e_3	Output
1	2	3	4	1	2	3	4	3	0	←	−1	1234
												4321
												2341
												1432
												3412
												2143
												4123
												3214
1	3	2	4	1	3	2	4	2	0	←	−1	1324
												4231
												3241
												1423
												2413
												3142
												4132
												2314
3	1	2	4	3	1	2	4	1	1	←	−1	3124
												4213
												1243
												3421
												2431
												1342
												4312
												2134